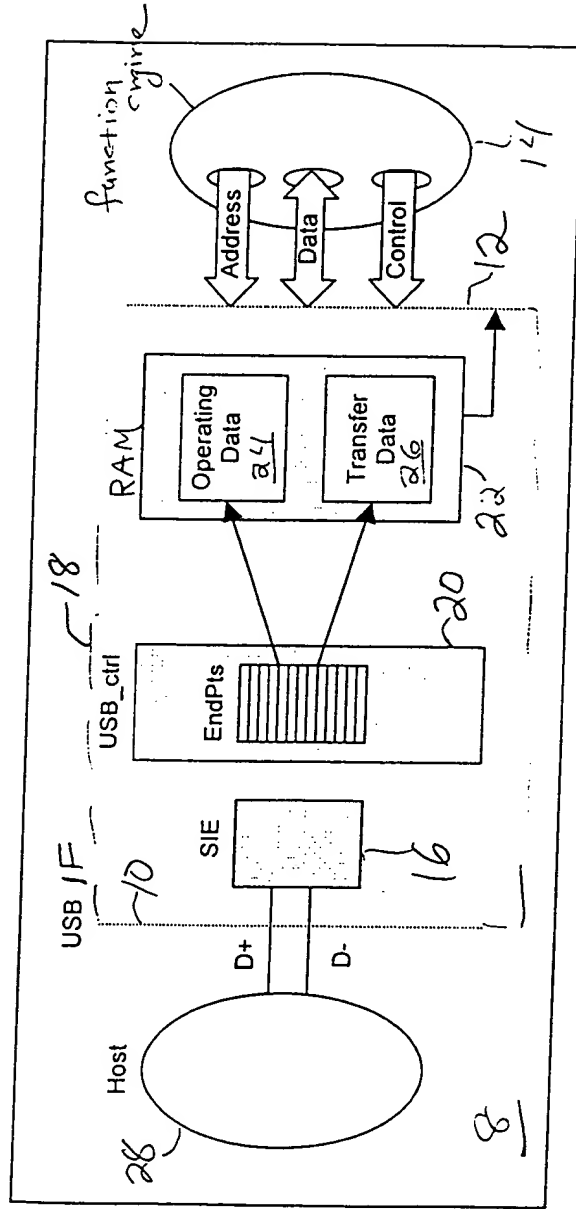


FIG. 1 is a block diagram of a USB device 100 according to an embodiment of the present invention. The USB device 100 includes a Host 28, a USB IF 10, a SIE 16, a USB\_ctrl 18, a RAM 22, and a function engine 14. The Host 28 is connected to the USB IF 10 via D+ and D- lines. The USB IF 10 is connected to the SIE 16. The SIE 16 is connected to the USB\_ctrl 18. The USB\_ctrl 18 is connected to the RAM 22. The RAM 22 is connected to the function engine 14. The function engine 14 includes an Address block 12, a Data block 12, and a Control block 12. The function engine 14 is connected to the Host 28 via the USB IF 10 and the SIE 16.



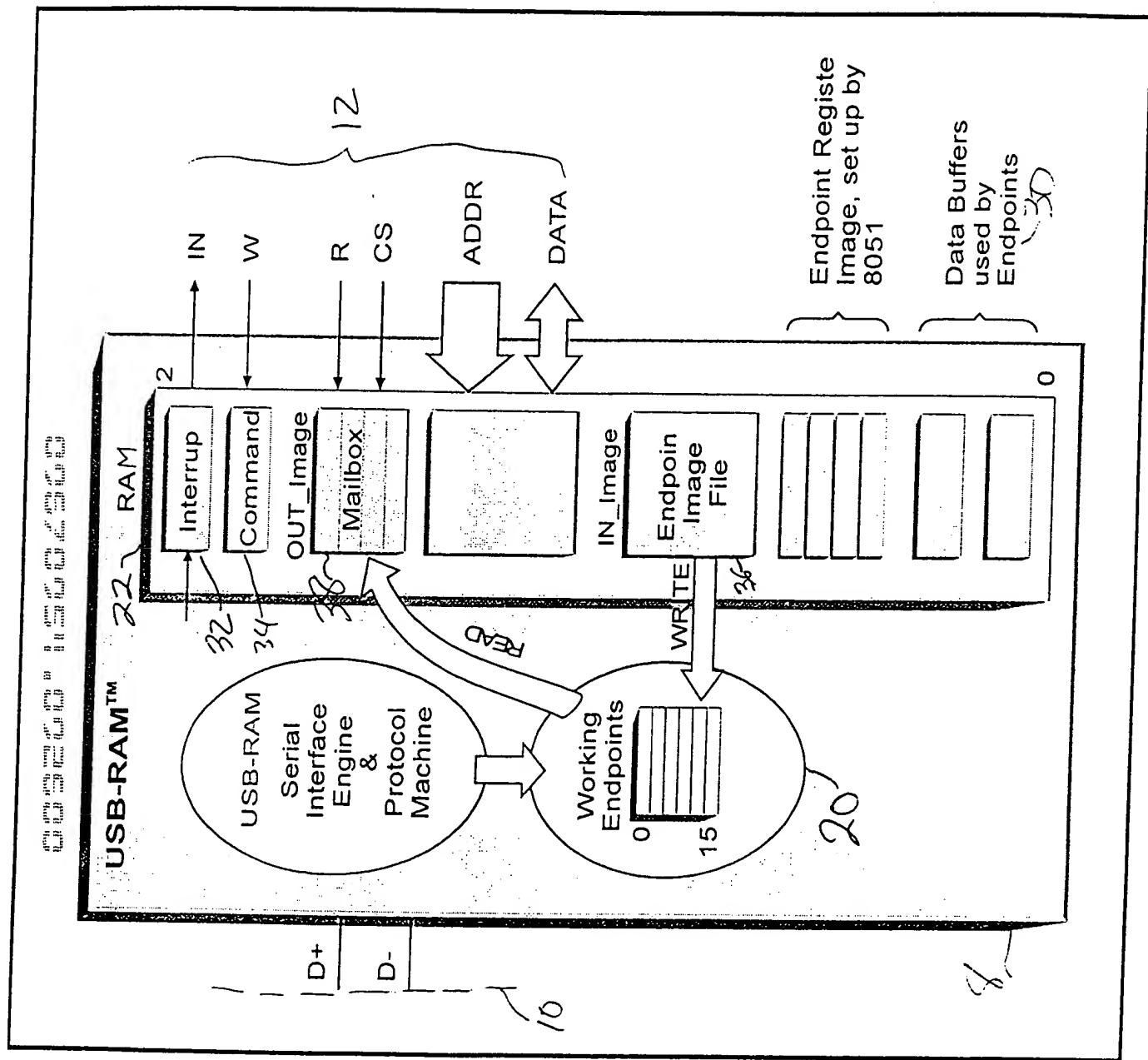
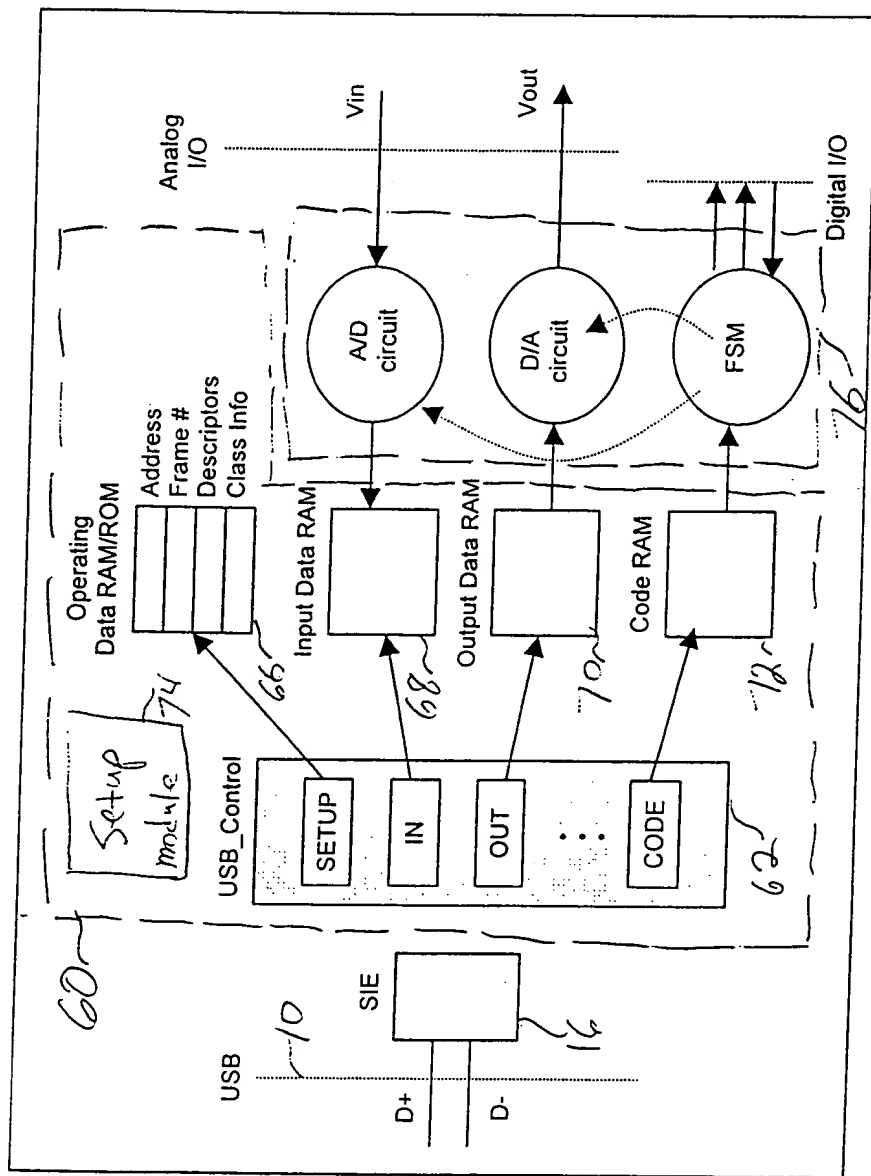
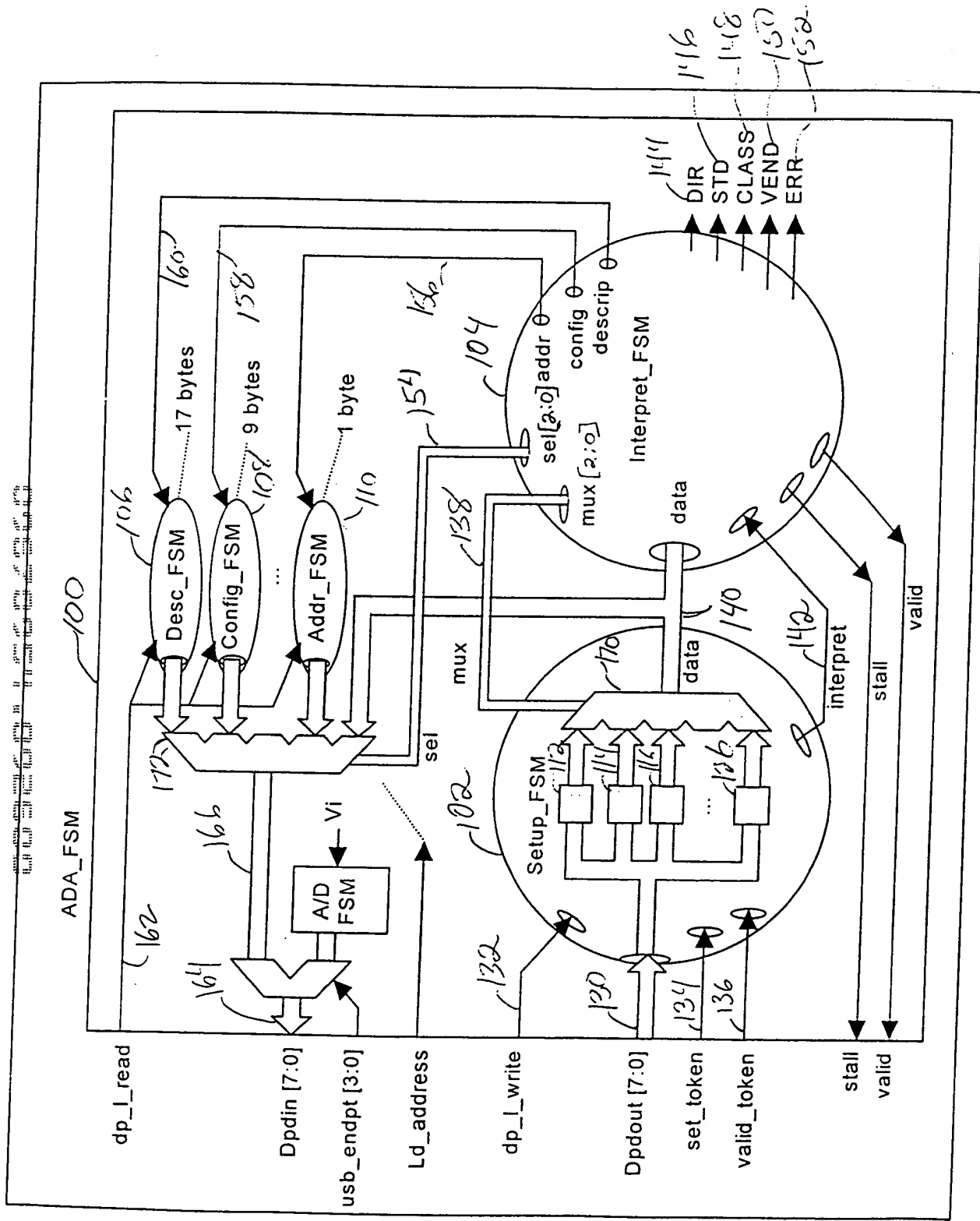


FIG. 2

Handwritten notes at the top of the page, possibly indicating a page number or reference.

Byte Count	Packet count	Valid	Type	Page#	Index
42	44	46	48	50	52





```

module ADA (clk,reset,dpdi,dpdout,usb_endpt,dp_l_read,
            dp_l_write,set_token,valid_token,stall,Vo,Vi)
output [7:0] dpdi;
output stall;
input [7:0] dpdout;
input [3:0] usb_endpt;
input set_token, valid_token, dp_l_read, dp_l_write;

SETUP_FSM

setup_FSM (.dpdout(dpdout),.clk (clk),.reset(reset),
.data (data),.mux (mux).dp_l_write(dp_l_write), .set_token
(set_token),.valid_token (valid_token));

INTERPRET_FSM

interpret_FSM(.clk (clk),.reset (reset), .data (data), .stall
(stall), .mux (mux),.sel (sel), .address (address), .config
(config), .descrip (descrip), .DIR (DIR), .STD (STD), .etc. );

DESCRIP_FSM

descrip_FSM (.d_data (d_data),.dp_l_read (dp_l_read), .descrip
(descrip),.reset (reset),.clk (clk));

CONFIG_FSM

config_FSM(.c_data (c_data),.dp_l_read (dp_l_read),
.config (config),.reset (reset),.clk (clk));

ADDRESS_FSM

address_FSM (.a_data (a_data),.dp_l_read (dp_l_read),
.address (address),.reset (reset),.clk (clk));

always @ (posedge clk) begin: Out_Mux:
case (sel)
    0 : s_data = d_data; // select descriptor data.
    1 : s_data = c_data; // select config data.
    2 : s_data = a_data; // select address data
    3 : s_data = other; // select other appropriate data.
endcase
end
wire [ 7:0 ] dpdi = (usb_endpt == 0) ? AD_out : s_data;
endmodule

```

FIG. 6A

```

module SETUP_FSM (dpdout, clk, reset, data, mux,
dp_l_write, set_token, valid_token);

input [7:0] dpdout;
input [3:0] usb_endpt;
input clk, reset, valid_token, set_token, dp_l_write;

output [7:0] data;
output interpret; reg interpret;

reg[7:0] REQUEST, REQ, VALUE1, VALUE2, INDEX1,
INDEX2, LENGTH1, LENGTH2;
reg[3:0] state, next_state;

always @ (negedge clk)
    if (set_token & valid_token) state = get_SETUP;
    else state = next_state;
always @ (posedge dp_l_write or posedge reset)
begin: SETUP_FSM
    if (reset) begin
        next_state = get_SETUP;
        interrupt <= 0;
    end
    else if (valid control endpoint and set_token)
begin case (state)
get_SETUP : begin
    interrupt <= 0;
    REQ[7] = dpdout[7];
    REQ[6:5] = {dpdout[6], dpdout[5]};
    next_state = get_REQ;
end
get_REQ : begin
    REQUEST = dpdout; // [7:0]
    next_state = get_VALUE1
end
get_VALUE1: begin
    VALUE1 = dpdout;
    next_state = get_VALUE2;
end
get_VALUE2: begin
    VALUE2 = dpdout;
    next_state = get_INDEX1;
end
get_INDEX1:
get_INDEX2:
get_LENGTH1: begin
    LENGTH1 = dpdout;
    next_state = get_LENGTH2;
end
get_LENGTH2: begin
    LENGTH2 = dpdout;
    next_state = get_LENGTH2;
    interpret <= 1; // enable interpret_FSM
end
endcase
endmodule

```

FIG. 6B

```

module INTERPRET_FSM (clk, reset, data,
stall, mux, sel, address, config,
descrip, DIR, STD, etc.);
input [7:0] data;
input interpret;
output [2:0] mux, sel;
output address, config, descrip; // or
other interpreted requests
output DIR, STD, etc; // CLASS, VENDOR,
etc.

```

```

reg address, config, descrip;
reg [2:0] state, next_state;

```

```

always @ (state or interpret or data)

```

```

begin : INTERPRET
if (interpret) begin

```

```

case (state)

```

```

INIT: begin
stall = 0;
mux = REQ_type;
next_state = got_REQ_type;
end

```

```

got_REQ_type: begin

```

```

DIR <= data[7];

```

```

case ({ data[6], data[5] })

```

```

00 : STD = 1;
01 : CLAS = 1;
10 : VEND = 1;
11 : ERR = 1;

```

```

endcase

```

```

next_state = get_REQ;

```

```

end

```

```

get_REQ: begin
mux = sel_REQ;
next_state = got_REQ;
end

```

```

got_REQ: begin

```

```

case (data)

```

```

00: begin
sel = `STATUS;
status = 1;
next_state = get_status;
end

```

```

05: begin
sel = `ADDRESS;
address = 1;
next_state = get_address
end

```

```

06: begin
sel = `DESCRIP;
descrip = 1;
next_state = get_desc
end

```

```

08: begin
sel = `CONFIG;
do_config = 1;
next_state = get_config;
end

```

```

0A: begin
sel = `INTERFACE;
interface = 1;
next_state = get_interface;
end

```

```

default: stall = 1;

```

```

endcase //data
end

```

```

get_address : begin
mux = VALUE2;
if (interpret) next_state = get_address;
else next_state = INIT;
end

```

```

endcase //state

```

```

end

```

```

always @ (posedge clk) begin
if(reset) state = INIT;
else state = next_state;
end

```

```

endmodule

```



```

Module  DESCRIP_FSM (desc, descrip, dp_1_read)

output [7:0] desc;
input          descrip, dp_1_read,

always @ (posedge dp_1_read) begin: descrip_FSM
if (descrip) begin
case (state)

0: begin
desc = length; //18
state = 1;
end

1: begin
desc = type; // = 1
state = 2;
end

2: begin
desc = USB. version; //1
state = 3;
end

3: begin
desc = next_byte, etc.
state = 4;
end

states 4 thru 16 not shown

17: begin
desc = last byte
state = last_state;
end

last state: begin . . .

endcase
end

endmodule

```

FIG. 6D

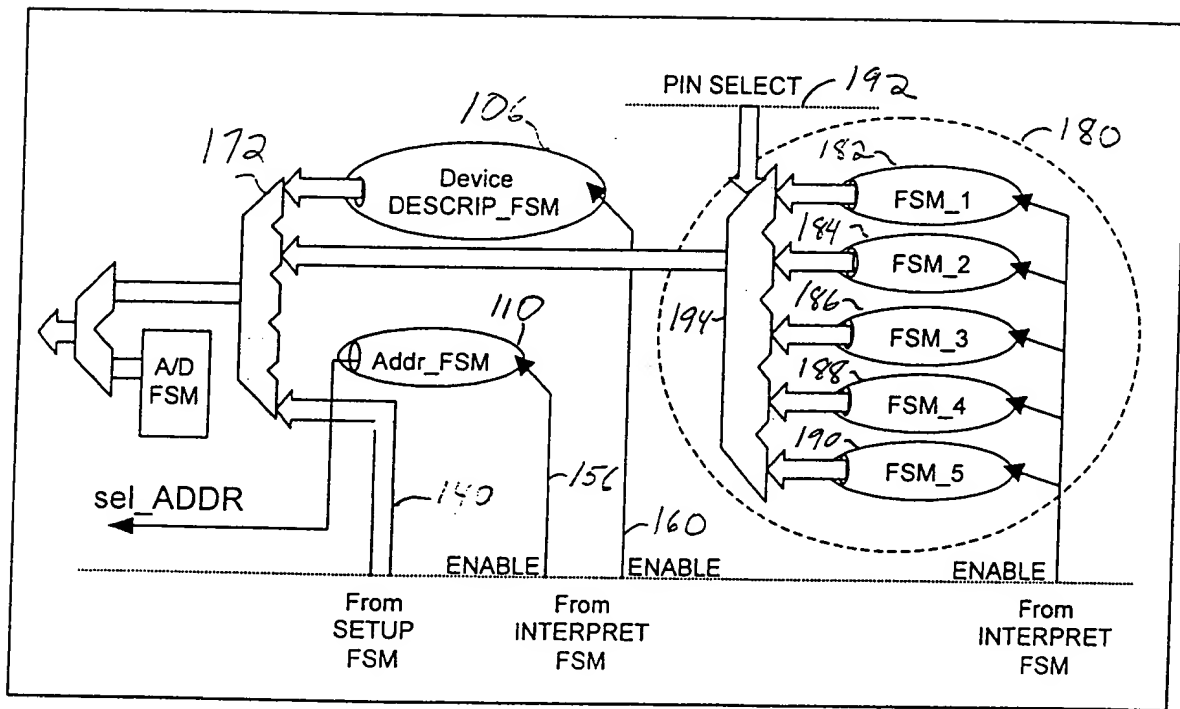


FIG. 7

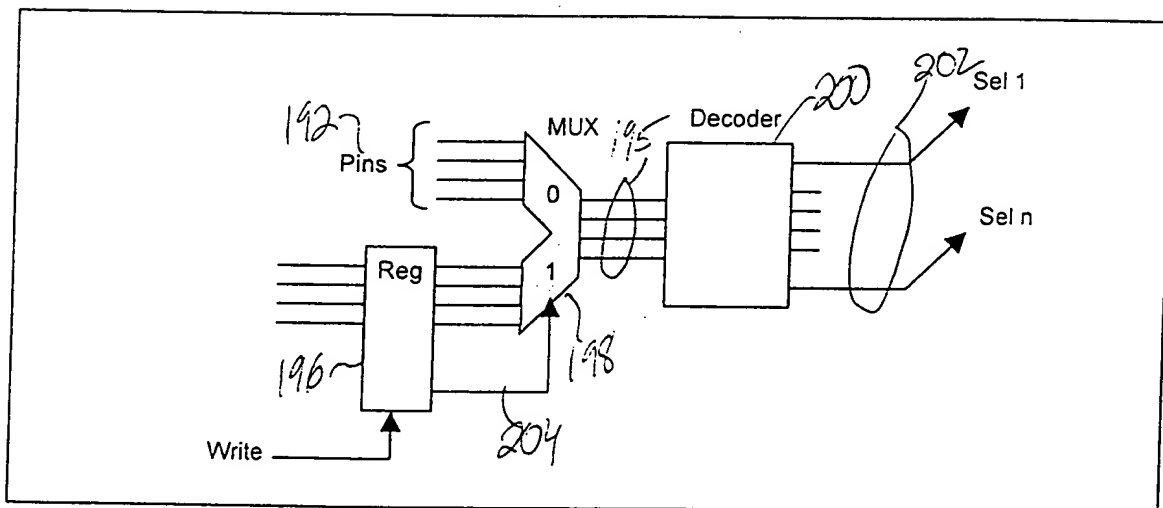


FIG. 8

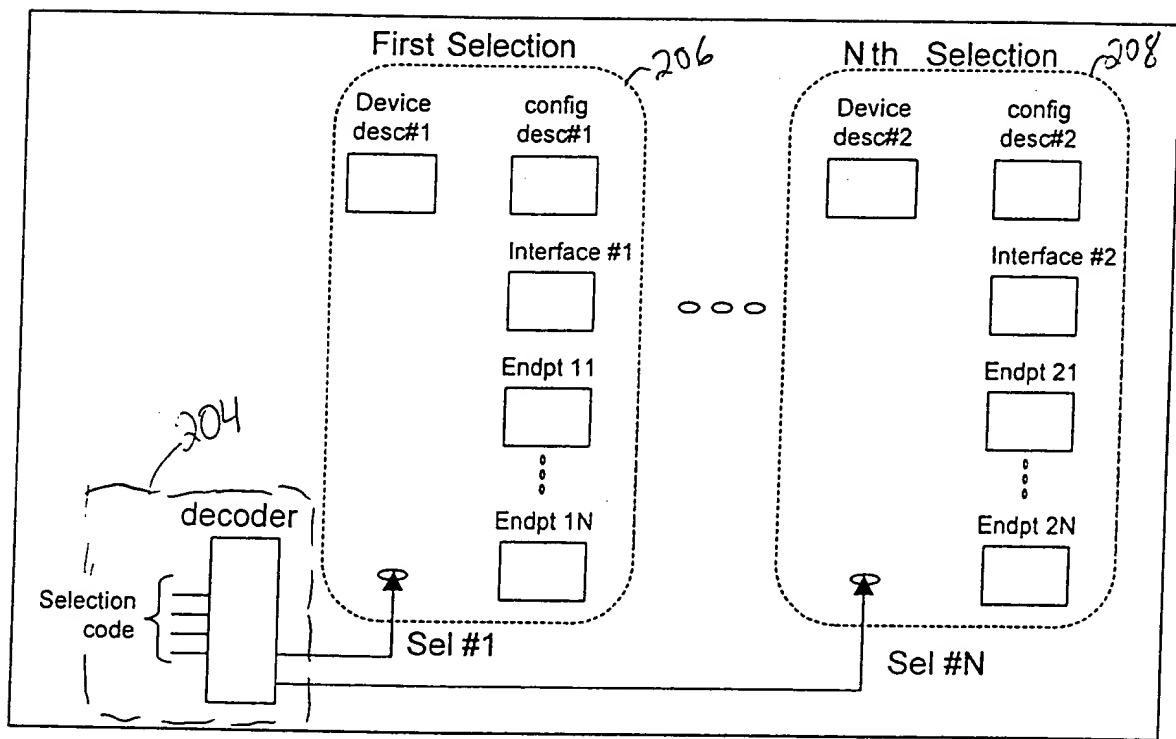


FIG. 9

